IFW

OIPE JC35 DEC 2 8 2004 PATENT & TRADEMARK OFFICE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:	§	Group Art Unit: 2151
	§	
Allavarpu, et al.	§	Examiner: Dinh, Khanh Q.
	§	
	§	Atty. Dkt. No.: 5181-46200
	§	P4466
Serial No. 09/552,985	§	
Filed: April 21, 2000	§	
For: CORBA Metadata Gateway to Telecommunications Management Network	§	

CERTIFICATE OF MAILING
37 C.F.R. § 1.8

I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as First Class Mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on the date indicated below:

Robert C. Kowert
Name of Registered Representative

December 21, 2004
Date

Signature

APPEAL BRIEF

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir/Madam:

Further to the Notice of Appeal filed October 26, 2004, Appellants present this Appeal Brief. Appellants respectfully request that the Board of Patent Appeals and Interferences consider this appeal.

12/30/2004 AWONDAF1 00000073 501505 09552985
01 FC:1402 500.00 DA

I. REAL PARTY IN INTEREST

As evidenced by the assignment recorded at Reel 010993, Frame 0870, the subject application is owned by Sun Microsystems, Inc., a corporation organized and existing under and by virtue of the laws of the State of Delaware, and now having its principal place of business at 4150 Network Circle, Santa Clara, CA 95054.

II. RELATED APPEALS AND INTERFERENCES

No other appeals, interferences or judicial proceedings are known which would be related to, directly affect or be directly affected by or have a bearing on the Board's decision in this appeal.

III. STATUS OF CLAIMS

Claims 1-31 are pending and rejected. The rejection of claims 1-31 is being appealed. A copy of claims 1-31 as on appeal is included in the Claims Appendix hereto.

IV. STATUS OF AMENDMENTS

No amendments to the claims have been submitted subsequent to the final rejection.

V. SUMMARY OF CLAIMED SUBJECT MATTER

Many types of devices may be managed over a network, such as printers, scanners, phone systems, copiers, and many other devices and appliances configured for network operation. Typically, network management software manages a given device by storing and manipulating a representation of its pertinent data as a software object, herein referred to as a "managed object." This managed object is the virtual representation of the device on the network. (*See, e.g.,* Specification, page 2, lines 1-16). In the prior art, network management frameworks provide information modeling standards to model

information about managed network devices (*See, e.g.*, Specification page 7, lines 7-17). Metadata may be used for understanding information related to managed devices, as well as information stored in data warehouses. Metadata may be data about a class, attributes, and/or other features of a managed object class. Metadata browser applications traditionally browse through modeling information to present information about various managed objects in the network. In prior art solutions, metadata browser applications use programming-language-specific, platform-specific, and hardware-specific APIs to access and display metadata information. This makes it difficult for such applications to interoperate with other network management applications that serve various network management functions. (*See, e.g.,* Specification page 7, lines 7-17).

Independent claim 1 is directed to a method for managing a network using a metadata gateway. A client may generate a request for type information for an attribute or event. The request may be expressed in an interface definition language operable to define object interfaces across a plurality of platforms and across a plurality of programming language. The request of type information may be sent to an object request broker (*See, e.g.*, FIG. 2, page 16, line 21 – page 17, line 6). Metadata may be retrieved through the metadata gateway by a client manager application sending a request for type information for a managed object attribute or event in IDL through a CORBA Object Request Broker (ORB) to the metadata gateway, which then reads the type information from a metadata repository, where the type information is stored in a database format. For example, as described at page 16, line 21 – page 17, line 23 of the Specification, manager applications 206 may be communicably coupled to a CORBA Object Request Broker (ORB) and may send IDL request and receive IDL responses through the ORB (see also, FIG. 6, page 23, lines 19-30, page 10, line 14- 22).

The metadata gateway may receive the request from type information from the object request broker. The metadata gateway provides translation of metadata to and from a database format and Interface Definition Language (IDL), which is operable across a plurality of platforms and across a plurality of programming languages (*See, e.g.*, FIG.

3, page 8, lines 15-21, page 17, line 27 – page 18, line 21). Type information may be read from a metadata repository storing the type information in a database format may be translated from the database format to the interface definition language. (*See, e.g.*, page 8, lines 3-13, page 18, line 5 – 21, page 19, lines 11-27). The metadata gateway then translates the retrieved type information from the database format to IDL and sends the translated type information to the ORB, which sends the translated type information for the attribute or event to the client manager application. The metadata gateway may then send the translated type information to the object request broker and the client may receive the translated type information, expressed in the interface definition language, for the attribute or event through the object request broker. (*See, e.g.*, FIG. 6, page 23, line 19- page 24, line 2).

Independent claim 10 is directed to a method or managing a network. As recited in claim 10, a client may generate a request to encode type information for an object, attribute, or event, wherein the request is expressed in an interface definition language, wherein the interface definition language is operable to define object interfaces across a plurality of platforms and across a plurality of programming languages (*See, e.g.*, page 10, line 25 – page 11, line 2, page 18, lines 11-21, FIG. 4, page 20, line 23-26, FIG. 7, page 24, lines 6-13). Claim 10 further recites sending the request to an object request broker and a metadata gateway receiving the request to encode the type information from the object request broker (*See, e.g.*, FIG. 3, page 17, line 29 – page 18, line 9). Additionally, the type information may be translated from the interface definition language to a database format and may be stored in a metadata repository in a database format (*See, e.g.*, FIG. 7, page 18, lines 11-21, page 24, line 21-29). Thus, metadata may be encoded through the metadata gateway by sending the metadata in IDL to the metadata gateway, which translates the type information from IDL to a database format and stores the type information in the metadata repository.

Independent claim 14 is directed to a network management system comprising a metadata repository and a metadata gateway. The metadata repository may include

metadata, expressed in a database format, concerning object classes for managed objects (*See, e.g.*, page 8, lines 9-13, page 23, lines 20-12). The metadata gateway may be coupled to the metadata repository and to an object request broker (*See, e.g.*, FIG. 6, page 23, lines 19-30, page 10, line 14- 22, 16, line 21 – page 17, line 23).

The metadata gateway may send and receive metadata from the database and may provide translation of the metadata to and from the database format and an interface definition language operable to define object interfaces across a plurality of platforms and across a plurality of programming languages (*See, e.g.*, page 8, lines 3-13, page 18, line 5 – 21, page 19, lines 11-27, FIG. 7, page 18, lines 11-21, page 24, line 21-29).

Independent claim 22 is directed to a carrier medium comprising program instructions that are computer-executable to implement a metadata gateway receiving a request for type information from an object request broker (*See, e.g.*, the discussion above regarding claim 1, FIGs. 3 and 6, page 8, lines 15-21, page 17, line 27 – page 18, line 21, and page 23, lines 19-30). The program instructions also implement reading the type information from a metadata repository, wherein the type information is stored in a database format in the metadata repository (*See, e.g.*, page 8, line 7 – page 9, line3, page 10, line 14- 22). Claim 22 further recites translating the type information from the database format to an interface definition language, and the metadata gateway sending the translated type information to the object request broker (*see, e.g.*, page 18, line 11-21, page 19, line 18-27). Further discussion and example embodiments regarding the limitations of claim 22 are discussed above regarding claim 1.

Independent claim 27 is directed to a carrier medium comprising program instructions that are computer-executable to implement a metadata gateway receiving a request to encode type information from an object request broker (*see, e.g.*, page 10, line 25 – page 11, line 2, page 18, lines 11-21, FIG. 4, page 20, line 23-26, FIG. 7, page 24, lines 6-13). The program instructions are also computer-executable to implement translating the type information from an interface definition language to a database format

and storing the type information in a metadata repository using a database format (*See, e.g.*, FIG. 3, page 17, line 29 – page 18, line 9, FIG. 7, page 18, lines 11-21, page 24, line 21-29).

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

1. Claims 1, 5, 7-10, 13-16, 18-22, 26, 27 and 31 are rejected under 35 U.S.C. § 102(e) as being anticipated by Schofield et al. (U.S. Pat. No. 6,263, 485).

2. Claims 2-4, 6, 11, 12, 17, 23-25 and 28-30 are rejected under 35 U.S.C. § 103(a) as being unpatentable over Schofield et al. (U.S. Pat. No. 6,263, 485) in view of Kulkarni, et al (U.S. Pat. No. 5,848,243).

VII. ARGUMENT

First Ground of Rejection:

Claims 1, 5, 7-10, 13-16, 18-21, 26, 27 and 31 are finally rejected under 35 U.S.C. § 102(e) as being anticipated by Schofield et al. (U.S. Pat. No. 6,263, 485) (hereinafter "Schofield"). Appellants traverse this rejection for at least the following reasons. Different groups of claims are addressed under their respective subheadings.

Claims 1, 7 and 9:

Schofield fails to teach a client generating a request *for* type information for an attribute or event, wherein the *request is expressed in an interface definition language*. Instead, Schofield teaches converting an original IDL description into new ASCII string descriptors that are "contained in a header file that is linked into both the client and server applications" (Schofield, column 3, lines 59-62, and column 11, lines 24-26). Schofield teaches a method for defining Interface Definition Language-defined data types, operations, or interfaces by generating an ASCII string descriptor that identifies the data type, interface, or operation (Schofield, Abstract). Specifically, Schofield teaches a code

generator 112 operable to generate files in the language of the client and server applications, and to produce a Compact IDL Notation (CIN) (Schofield, col. 6, lines 23 – 51). Additionally, Schofield teaches the benefits of using these ASCII strings instead of IDL to describe operations and data types (Schofield, column 4, lines 5-11). Thus, **Schofield teaches away** from using IDL to express requests.

Furthermore, the Examiner's interpretation of IDL source file 101 in FIG. 4 as a client that can generate a request for type information is clearly incorrect. Schofield describes IDL source file 101 as a source file containing IDL interface definitions that is compiled by IDL compiler 103. A source file cannot be a client and cannot generate anything.

In addition, Schofield does not teach a client generating a request *for type information*. As the Examiner states in the Response to Arguments section of the Final Office Action, Schofield teaches that client requests *include* data type definitions so that a capsule can create and/or load an appropriate object according to a received request (Schofield, col. 6, lines 1-12). Appellants note that in the Response to Arguments section the Examiner asserts that Schofield teaches generating requests *for* type information, but Schofield clearly describes requests *including* type information, not requests *for* type information. These are two very different types of requests. A request that already *includes* type information is clearly not a request *for* type information. In Schofield, the requester already has type information and includes that type information in client requests for object operations. In fact, central to Schofield's system is a method for compiling IDL source files that contain "interface definitions, operation definitions, and individual data type definitions" (Schofield, col. 7, lines 11-14) into client and server applications (See also, Schofield, col. 6, lines 13-49). Thus, Schofield clearly fails to teach a client generating a request ***for*** type information.

In further regard to claim 1, Schofield does not teach sending the request for type information to an object request broker. There is no teaching anywhere in Schofield

regarding sending a request for type information to an object request broker. Schofield's invention is directed to compiling client and server stub interfaces based on type definitions from IDL source 101. Both client and server applications in Schofield's system contain type correct and language-specific request routines for communicating with each other (See, Schofield, column 3, lines 13-29). As such, clients in Schofield's system already has type correct method calls and therefore have no need to send requests for type information to an object request broker. Hence, Schofield does not anticipate sending a request for type information to an object request broker.

The Examiner has incorrectly interpreted Schofield's code generator 112 as an object request broker. The Examiner also argues that Schofield's IDL source 101 is a client that sends a request for type information to code generator 112 and server stub 89. Compiling a source file and using it to create additional source files to then be compiled into application and implementation libraries is not a client sending a request for type information to an object request broker. Furthermore, since the compiled IDL source 101 is used to create server stub 89, it does not make any sense that IDL source 101 can send a request for type information to another set of source files. Schofield's code generator 112 is operable to generate files in the language of the client and server applications, and to produce a Compact IDL notation as illustrated in Figs. 5 – 7 and described starting at col. 7, line 20. Schofield does not teach that code generator 112 receives requests for type information as asserted by the Examiner. Schofield is very clear that code generator 112 "parses a source file line by line" and "produces an ASCII descriptor in the header file to be included by the client and server applications" (Schofield, col. 6, lines 50-64). The concept of an object request broker is well known in the art and Schofield's code generator is not an object request broker as one of ordinary skill in the art would recognize.

Schofield further fails to anticipate a metadata gateway receiving the request for type information from the object request broker. As argued above, Schofield fails to teach sending a request for type information to an object request broker. For that reason

alone, Schofield also fails to teach a metadata gateway receiving such a request from an object request broker. Additionally, Schofield makes no mention of any metadata gateway. Nor does Schofield describe a metadata gateway receiving a request for type information from an object request broker.

The Examiner argues that Schofield's server stub 89 is a metadata gateway. Appellants, however, disagree with such an interpretation of Schofield. Schofield teaches that "code generator 112 produces a client stub file 79 containing client stub functions and a server stub file 89 containing definitions for object implementations" (Schofield, col. 6, lines 30 – 32). Schofield does not teach that server stub 89 receives requests for type information from code generator 112, which the Examiner views as an object request broker. In fact, Schofield describes how code generator 112 creates server stub 89. Once Schofield's system is running (and requests are generated and sent) code generator 112 would not even be executing. Code generator 112 is only used to compile portions of client and server applications. Appellants fail to see how code generate 112 can be considered a metadata gateway that receives a request for type information from an object request broker.

The Examiner, in the Advisory Action of October 21, 2004, argues that server stub 89 receives a request for type information from code generator 112. However, Schofield specifically describes how code generator 112 parses a PIF file and generates client and server stub files containing stub functions. Code generator 112 also creates a header that is included in the client and server stubs. Nowhere does Schofield mention code generator 112 sending a request for type information to a client stub. Furthermore, since code generator 112 is used only to create the sources files that will be compiled into server stub 89, and since code generator 112 and server stub 89 are never executed at the same time, there would be no way for code generator 112 to send a request to server stub 89.

Schofield also fails to anticipate reading type information from a metadata repository, wherein the type information is stored in a database format in the metadata repository. Instead, as described above, Schofield teaches the compiling of type specific client and server routines into implementation libraries. Nowhere does Schofield store type information in a database format in a metadata repository. There is no need in Schofield's system for reading type information from a metadata repository since the implementation libraries already contain type correct methods and routines.

Appellants further disagree with the Examiner's characterization of implementation library 81 from Fig. 4 as a metadata repository. Schofield teaches that the implementation library includes executable code such as the server applications and server stubs generated from the IDL specification of the object's interface (Schofield, column 5, lines 55-59). It is well understood that a library of executable code is not a metadata repository. Schofield clearly describes how a compiler links the server application object code and the server stub object code to produce implementation library 81 (Schofield, col. 6, lines 47-49). Thus implementation library 81 contains server routines in executable format and cannot be a metadata repository from which type information is read and that stores the type information in a *database* format.

In the Advisory Action, the Examiner argues that Schofield uses client stub object code to produce a metadata repository. This is clearly the Examiner's own hindsight-based speculation since it is not mentioned at all by Schofield. Instead, Schofield teaches that client stub object code is linked to produce implementation library 71 and similarly server stub object code is linked to produce implementation library 81 (Schofield, column 6, lines 34-59). As argued above, implementation libraries 71 and 81 are libraries of executable routines and not metadata repositories.

Schofield also fails to teach translating the type information *from the database format to the interface definition language*. Firstly, as shown above, Schofield does not teach storing type information in a database format, but rather compiles header files for

inclusion in client and server applications. Secondly, Schofield describes generating a CIN (ASCII based) descriptor ***from*** an IDL data type (Schofield, column 7, lines 20-22). In his Response to Arguments, the Examiner incorrectly states Appellants' argument. Appellants actually argue (as previously presented) that Schofield fails to teach translating the type information ***from*** *the database format* ***into*** *the interface definition language*. Instead, Schofield describes his method for generating a CIN (ASCII based) descriptor ***from*** an IDL data type (Schofield, column 7, lines 20-22). In fact, Schofield clearly illustrates this with an example where an ADD operation is translated from its IDL descriptor, "long Add (in long x, in long y);" to its equivalent CIN descriptor, "126861413+3+ADDA3+DFAFAF0+0+" (Schofield, column 10, lines 41-51). Hence, **Schofield teaches the opposite** of translating the type information ***into*** the interface definition language. In the Advisory Action, the Examiner again misrepresents Appellants' argument. The Examiner assumes that Appellants are arguing that Schofield fails to teach "translating type information in IDL." However, as described above, Appellants are actually arguing that Schofield fails to teach translating type information from a database format into an interface definition language and that Schofield actually teaches the opposite. Thus, the Examiner's arguments regarding Schofield compiling IDL source files fails to rebut Appellants' argument.

Further, Appellants disagree with the Examiner's assertion that Schofield teaches "the client receiving the translated type information for the attribute or event through the object request broker, wherein the translated type information is expressed in the interface definition language." As shown above, Schofield fails to teach translating type information into IDL. Thus, any **translated type information** in Schofield is expressly taught as **not in IDL**, but in Schofield's ASCII based CIN descriptor language (See, Schofield, Abstract, col. 3, line 59 – col. 4, line 11, Fig. 5, and col. 7, line 20 – col. 11, line 36).

Claims 7 and 9 are patentable of Schofield for at least the reasons presented above regarding independent claim 1 from which they depend.

Claim 5:

Claim 5 is patentable over Schofield for at least the reasons given above in regard to claim 1 from which it depends. Furthermore, Schofield also fails to anticipate wherein the sending the request for type information to an object request broker, the metadata gateway receiving the request for type information from the object request broker, the metadata gateway sending the translated type information to the object request broker, and the client receiving the translated type information for the attribute or event through the object request broker are effected via an internet inter-object communication protocol.

The Examiner argues that Schofield uses IDL source 101 as a "translation unit" and cites Figure 4, column 3, line 8-column 4, line 12, and column 5, line 17 – column 6, line 65. However, Figure 4 of Schofield clearly illustrates a compilation process that occurs on a single computer and therefore has no need to communicate using an internet inter-object communication protocol (see, Schofield, column 6, lines 13-20). The Examiner's other cited passage also describes the compiling of server applications, client applications and implementation libraries. The Examiner interprets Schofield's IDL source 101 as a client that requests type information from code generator 112, which the Examiner maintains is an object request broker and that server stub 89 is a metadata gateway that communicates with implementation library 81, which the Examiner contends is a metadata repository. Essentially every aspect of the Examiner's argument is faulty. Please refer Appellants' arguments regarding claim 1 above for a more detailed explanation of the Examiner's incorrect interpretation of Schofield.

Fig 4 of Schofield and the accompanying description (column 6, line 13 – column 7, line 17) also make it very clear that Schofield is teaching a process for compiling various server and object routines in both stub applications and implementation libraries so that an appropriate object can be created when necessary to facilitate communication between clients and servers. (See, Schofield, Summary of the Invention, column 3, line

54 – column 4, line 19). Thus, the compilation process, which the Examiner erroneously interprets as a client requesting and receiving type information that is retrieved and translated by a metadata gateway and delivered via an object request broker, actually has no need whatsoever for communicating via an internet inter-object communication protocol.

Claim 8:

Claim 8 is patentable over Schofield for at least the reasons above regarding claim 1 from which it depends. Additionally, Schofield fails to anticipate wherein the metadata gateway is distributed over a plurality of servers, wherein each of the plurality of servers presents a substantially identical view of the metadata gateway. The Examiner has (incorrectly) interpreted Schofield's server stub 89 as a metadata gateway. Schofield does not describe server stub 89 as distributed over a plurality of servers wherein each server presents a substantially identical view of server stub 89. In contrast, Schofield teaches that code generator 112 creates server stub file 89 containing definitions for object implementations and that server stub file 89 is compiled by programming language-specific compiler 124 "to produce … compiled server stub object code." The server stub object code is compiled to create implementation library 81. (See, FIG. 4 and column 6, lines 30-37 and lines 45-49). Thus, server stub 89, which the Examiner contends is a metadata gateway, is actually executable object code generated as part of creating server applications and server implementation libraries. Schofield does not describe distributing server stub object code across multiple servers, nor would it make any sense to do so.

Furthermore, even using a more reasonable interpretation of Schofield's teachings, Schofield still fails to anticipate wherein the metadata gateway is distributed over a plurality of servers, wherein each of the plurality of servers presents a substantially identical view of the metadata gateway. There is no mention in Schofield of any metadata gateway distributed over a plurality of servers that functions as recited in Appellants' claim. Schofield only describes how clients and servers may be executed on separate machines (see, Schofield, FIG. 1, column 5, lines 9-15, and lines 22-29).

Claim 10:

Schofield fails to teach a client generating a request to encode type information for an object, attribute or event, wherein the *request is expressed in an interface definition language*. Instead, Schofield teaches converting an original IDL description into new ASCII string descriptors that are "contained in a header file that is linked into both the client and server applications" (Schofield, column 3, lines 59-62, and column 11, lines 24-26). Schofield teaches a method for defining Interface Definition Language-defined data types, operations, or interfaces by generating an ASCII string descriptor that identifies the data type, interface, or operation (Schofield, Abstract). Specifically, Schofield teaches a code generator 112 operable to generate files in the language of the client and server applications, and to produce a Compact IDL Notation (CIN) (Schofield, col. 6, lines 23 – 51). Additionally, Schofield teaches the benefits of using these ASCII strings instead of IDL to describe operations and data types (Schofield, column 4, lines 5-11). Thus, **Schofield teaches away** from using IDL to express requests.

Additionally, the Examiner has erroneously interpreted Schofield's IDL source 101 as a client that generates a request to encode type information. For more information regarding this argument, please refer to the discussion of IDL source 101 above regarding claim 1.

In addition, Schofield does not teach a client generating a request *to encode type information*. Instead, Schofield teaches using an IDL compiler to compile client and server stub object code using predefined IDL-based interface definitions (Schofield, column 3, lines8-28). Thus, any client in Schofield's system has all necessary type information hard coded in the appropriate source files (See also, FIG 4, and column 6, lines 13-49).

In further regard to claim 10, Schofield does not teach sending the request for type information to an object request broker. There is no teaching anywhere in Schofield

regarding sending a request for type information to an object request broker. Schofield's invention is directed to compiling client and server stub interfaces based on type definitions from IDL source 101. Both client and server applications in Schofield's system contain type correct and language-specific request routines for communicating with each other (See, Schofield, column 3, lines 13-29). As such, clients in Schofield's system already has type correct method calls and therefore have no need to send requests for type information to an object request broker. Hence, Schofield does not anticipate sending a request for type information to an object request broker.

The Examiner has incorrectly interpreted Schofield's code generator 112 as an object request broker. The Examiner also argues that Schofield's IDL source 101 is a client that sends a request for type information to code generator 112 and server stub 89. Compiling a source file and using it to create additional source files to then be compiled into application and implementation libraries is not a client sending a request for type information to an object request broker. Furthermore, since the compiled IDL source 101 is used to create server stub 89, it does not make any sense that IDL source 101 can send a request for type information to another set of source files. Schofield's code generator 112 is operable to generate files in the language of the client and server applications, and to produce a Compact IDL notation as illustrated in Figs. 5 – 7 and described starting at col. 7, line 20. Schofield does not teach that code generator 112 receives requests for type information as asserted by the Examiner. Schofield is very clear that code generator 112 "parses a source file line by line" and "produces an ASCII descriptor in the header file to be included by the client and server applications" (Schofield, col. 6, lines 50-64). The concept of an object request broker is well known in the art and Schofield's code generator is not an object request broker as one of ordinary skill in the art would recognize.

Schofield further fails to anticipate a metadata gateway receiving the request for type information from the object request broker. As argued above, Schofield fails to teach sending a request for type information to an object request broker. For that reason

alone, Schofield also fails to teach a metadata gateway receiving such a request from an object request broker. Additionally, Schofield makes no mention of any metadata gateway. Nor does Schofield describe a metadata gateway receiving a request for type information from an object request broker.

The Examiner argues that Schofield's server stub 89 is a metadata gateway. Appellants, however, disagree with such an interpretation of Schofield. Schofield teaches that "code generator 112 produces a client stub file 79 containing client stub functions and a server stub file 89 containing definitions for object implementations" (Schofield, col. 6, lines 30 – 32). Schofield does not teach that server stub 89 receives requests for type information from code generator 112, which the Examiner views as an object request broker. In fact, Schofield describes how code generator 112 creates server stub 89. Once Schofield's system is running (and requests are generated and sent) code generator 112 would not even be executing. Code generator 112 is only used to compile portions of client and server applications. Appellants fail to see how code generate 112 can be considered a metadata gateway that receives a request for type information from an object request broker.

The Examiner, in the Advisory Action of October 21, 2004, argues that server stub 89 receives a request for type information from code generator 112. However, Schofield specifically describes how code generator 112 parses a PIF file and generates client and server stub files containing stub functions. Code generator 112 also creates a header that is included in the client and server stubs. Nowhere does Schofield mention code generator 112 sending a request for type information to a client stub. Furthermore, since code generator 112 is used only to create the sources files that will be compiled into server stub 89, and since code generator 112 and server stub 89 are never executed at the same time, there would be no way for code generator 112 to send a request to server stub 89.

Schofield also fails to anticipate storing type information in a metadata repository, wherein the type information is stored in a database format in the metadata repository. Instead, as described above, Schofield teaches the compiling of type specific client and server routines into implementation libraries. Nothing in Schofield's system can be considered a metadata repository and nowhere does Schofield describe storing type information in a database format in a metadata repository.

Appellants further disagree with the Examiner's characterization of implementation library 81 from Fig. 4 as a metadata repository. Schofield teaches that the implementation library includes executable code such as the server applications and server stubs generated from the IDL specification of the object's interface (Schofield, column 5, lines 55-59). It is well understood that a library of executable code is not a metadata repository. Schofield clearly describes how a compiler links the server application object code and the server stub object code to produce implementation library 81 (Schofield, col. 6, lines 47-49). Thus implementation library 81 contains server routines in executable format and cannot be a metadata repository from which type information is read and that stores the type information in a *database* format.

In the Advisory Action, the Examiner argues that Schofield uses client stub object code to produce a metadata repository. This is clearly the Examiner's own hindsight-based speculation since it is not mentioned at all by Schofield. Instead, Schofield teaches that client stub object code is linked to produce implementation library 71 and similarly server stub object code is linked to produce implementation library 81 (Schofield, column 6, lines 34-59). As argued above, implementation libraries 71 and 81 are libraries of executable routines and not metadata repositories.

Claim 13:

Claim 13 is patentable over Schofield for at least the reasons given above in regard to claim 10 from which it depends. Furthermore, Schofield also fails to anticipate wherein the sending the request to an object request broker and the metadata gateway

receiving the request to encode the type information from the object request broker are effected via an internet inter-object communication protocol.

The Examiner argues only that Schofield uses IDL source 101 as a "translation unit" and cites Figure 4, column 3, line 8-column 4, line 12, and column 5, line 17 – column 6, line 65. However, Figure 4 of Schofield clearly illustrates a compilation process that occurs on a single computer and therefore has no need to communicate using an internet inter-object communication protocol (see, Schofield, column 6, lines 13-20). The Examiner's other cited passage also describes the compiling of server applications, client applications and implementation libraries. The Examiner interprets Schofield's IDL source 101 as a client that requests type information from code generator 112, which the Examiner maintains is an object request broker and that server stub 89 is a metadata gateway that communicates with implementation library 81, which the Examiner contends is a metadata repository. Essentially every aspect of the Examiner's argument is faulty. Please refer Appellants' arguments regarding claim 10 above for a more detailed explanation of the Examiner's incorrect interpretation of Schofield.

Fig 4 of Schofield and the accompanying description (column 6, line 13 – column 7, line 17) also make it very clear that Schofield is teaching a process for compiling various server and object routines in both stub applications and implementation libraries so that an appropriate object can be created when necessary to facilitate communication between clients and servers. (See, Schofield, Summary of the Invention, column 3, line 54 – column 4, line 19). Thus, the compilation process, which the Examiner erroneously interprets as a client requesting and receiving type information that is retrieved and translated by a metadata gateway and delivered via an object request broker, actually has no need whatsoever for communicating via an internet inter-object communication protocol.

Claims 14 and 21:

Regarding claim 14, Schofield does not teach a metadata gateway which is communicatively coupled to the metadata repository and to an object request broker, wherein the metadata gateway is operable to send and receive the metadata from the database, wherein the metadata gateway provides translation of the metadata to and from the database format and an interface definition language, wherein the interface definition language is operable to define object interfaces across a plurality of platforms and across a plurality of programming languages.

Appellants respectfully disagree with the Examiner's characterization of server stub 89 from Fig. 4 as a metadata gateway. Schofield teaches that "code generator 112 produces a client stub file 79 containing client stub functions and a server stub file 89 containing definitions for object implementations" (Schofield, col. 6, lines 30 – 32). Schofield does not teach that server stub 89 receives requests for type information from code generator 112, which the Examiner views as an object request broker. In fact, Schofield describes how code generator 112 creates server stub 89. Once Schofield's system is running (and requests are generated and sent) code generator 112 would not even be executing. Code generator 112 is only used to compile portions of client and server applications. Appellants fail to see how code generator 112 can be considered a metadata gateway that receives a request for type information from an object request broker.

The Examiner, in the Advisory Action, argues that server stub 89 receives a requests from code generator 112. However, Schofield specifically describes how code generator 112 parses a PIF file and generates client and server stub files containing stub functions. Code generator 112 also creates a header that is included in the client and server stubs. Nowhere does Schofield mention code generator 112 sending a request for type information to a client stub. Furthermore, since code generator 112 is used only to create the sources files that will be compiled into server stub 89, and since code generator 112 and server stub 89 are never executed at the same time, there would be no way for code generator 112 to send a request to server stub 89.

The Examiner erroneously interprets Schofield's code generator 112 as an object request broker. Schofield's code generator 112 is operable to generate files in the language of the client and server applications, and to produce a Compact IDL notation as illustrated in Figs. 5 – 7 and described starting at col. 7, line 20. Schofield does not teach that code generator 112 receives requests for type information as asserted by the Examiner. Schofield is very clear that code generator 112 "parses a source file line by line" and "produces an ASCII descriptor in the header file to be included by the client and server applications" (Schofield, col. 6, lines 50-64). The concept of an object request broker is well known in the art and Schofield's code generator is not an object request broker as one of ordinary skill in the art would recognize.

Appellants disagree with the Examiner's characterization of implementation library 81 from Fig. 4 as a metadata repository. Schofield teaches that the implementation library includes executable code such as the server applications and server stubs generated from the IDL specification of the object's interface (Schofield, column 5, lines 55-59). Schofield clearly describes how a compiler links the server application object code and the server stub object code to produce implementation library 81 (Schofield, col. 6, lines 47-49). Thus implementation library 81 contains server routines in executable format and thus cannot be a metadata repository from which type information is read and that stores the type information in a *database* format.

In the Advisory Action, the Examiner argues that Schofield uses client stub object code to produce a metadata repository. However this is clearly the Examiner's own hindsight-based speculation since it is not mentioned at all by Schofield. Instead, Schofield teaches that client stub object code is linked to produce implementation library 71 and similarly server stub object code is linked to produce implementation library 81 (Schofield, column 6, lines 34-59). As argued above, implementation libraries 71 and 81 are libraries of executable routines and not a metadata repository.

Claims 18, 19 and 21 are patentable over Schofield for at least the reasons presented above regarding claim 14 from which they depend.

Claim 15:

Claim 15 is patentable over Schofield for at least the reasons set forth above regarding claim 14, from which it depends. Additionally, Schofield fails to anticipate wherein the managed devices comprise a telephone system. The Examiner cites figures 3 and 4 of Schofield, as well as column 5, line 51 to column 7, line 18. However, the Examiner fails to point out what portion of Schofield's system the Examiner considers a telephone system. Schofield fails to mention a telephone system, either in the Examiner's cited passages or anywhere else. Neither figure 3, nor figure 4, illustrate anything that can be considered a telephone system. Specifically, figure 3 illustrates software components, none of which can be considered anything remotely resembling a telephone system. Figure 4, also illustrates various software related items, from source code to compiled executable modules. Again, none of the items illustrated in figure 3 or figure 4 are described by Schofield as having anything to do with a telephone system. The Examiner also cites column 5, line 51 through column 7, line 18, where Schofield the overall workings of his system and described in detail the compilation of IDL source files into server and client application executables. Nowhere does Schofield mention or suggest a telephone system.

Claim 16:

Claim 16 is patentable over Schofield for at least the reasons set forth above regarding claim 14, from which it depends. Additionally, Schofield fails to anticipate wherein the managed devices comprise a network switch. The Examiner again cites figures 3 and 4 of Schofield, as well as column 5, line 51 to column 7, line 18. However, the Examiner fails to point out what portion of Schofield's system the Examiner considers a network switch. Schofield fails to mention a network switch, either in the Examiner's cited passages or anywhere else. Neither figure 3, nor figure 4, illustrated

anything that can be considered a network switch. Specifically, figure 3 illustrates software components, none of which can be considered anything remotely resembling a network switch. Figure 4, also illustrates various software related items, from source code to compiled executable modules. Again, none of the items illustrated in figure 3 or figure 4 are described by Schofield as having anything to do with a network switch. The Examiner also cites column 5, line 51 through column 7, line 18, where Schofield the overall workings of his system and described in detail the compilation of IDL source files into server and client application executables. Nowhere does Schofield mention or suggest a network switch.

Claims 18 and 19:

Claims 18 and 19 are patentable over Schofield for at least the reasons presented above regarding claim 14 from which they depend. They are also patentable for at least the reasons presented below regarding claim 17, from which they also depend.

Furthermore, the rejection of claims 18 and 19 is improper because the Examiner has rejected claims 18 and 19 under 35 U.S.C. § 102(e) as being anticipated by Schofield, but they depend from claim 17, which the Examiner has rejected under 35 U.S.C. § 103(a). Thus, the § 102(e) anticipation rejection of claims 18 and 19 is improper.

Claim 20:

Claim 20 is patentable over Schofield for at least the reasons given above regarding claim 14, from which it depends. Furthermore, Schofield also fails to teach wherein the object request broker is configurable to be accessed by a plurality of network management clients to obtain the metadata as expressed in the generic interface. The Examiner cites column 7, line 20 to column 8, line 67 and column 10, lines 9-51. Both of the passages cited by the Examiner describes the compiling of IDL source 101 using CIN descriptors. The Examiner has previously (see rejection of claim 14) argued that Schofield's code generator 112 is an object request broker, but the Examiner fails to

explain how code generator 112 is configurable to be accessed by a plurality of network management clients. Schofield clearly describes how code generator 112 uses a PIF file and generates client and server stub source file that will be compiled into client and server implementation libraries. Schofield does not teach that network management clients can access that code generator 112. The Examiner has interpreted IDL source 101 as a client – but, as argued above, regarding claim 14, such an interpretation is clearly incorrect.

Claims 22 and 26:

Regarding claim 22, Schofield fails to anticipate <u>a metadata gateway receiving a request for type information from an object request broker</u>. Schofield makes no mention of any metadata gateway. Nor does Schofield describe a metadata gateway receiving a request for type information from an object request broker.

The Examiner argues that Schofield's server stub 89 is a metadata gateway. Appellants, however, disagree with such an interpretation of Schofield. Schofield teaches that "code generator 112 produces a client stub file 79 containing client stub functions and a server stub file 89 containing definitions for object implementations" (Schofield, col. 6, lines 30 – 32). Schofield does not teach that server stub 89 receives requests for type information from code generator 112, which the Examiner views as an object request broker. In fact, Schofield describes how code generator 112 creates server stub 89. Once Schofield's system is running (and requests are generated and sent) code generator 112 would not even be executing. Code generator 112 is only used to compile portions of client and server applications. Appellants fail to see how code generate 112 can be considered a metadata gateway that receives a request for type information from an object request broker.

The Examiner, in the Advisory Action of October 21, 2004, argues that server stub 89 receives a request for type information from code generator 112. However, Schofield specifically describes how code generator 112 parses a PIF file and generates client and server stub files containing stub functions. Code generator 112 also creates a

header that is included in the client and server stubs. Nowhere does Schofield mention code generator 112 sending a request for type information to a client stub. Furthermore, since code generator 112 is used only to create the sources files that will be compiled into server stub 89, and since code generator 112 and server stub 89 are never executed at the same time, there would be no way for code generator 112 to send a request to server stub 89. Furthermore, he Examiner has incorrectly interpreted Schofield's code generator 112 as an object request broker. Schofield's code generator 112 is operable to generate files in the language of the client and server applications, and to produce a Compact IDL notation as illustrated in Figs. 5 – 7 and described starting at col. 7, line 20. Schofield does not teach that code generator 112 sends requests for type information to a metadata gateway as asserted by the Examiner. Schofield is very clear that code generator 112 "parses a source file line by line" and "produces an ASCII descriptor in the header file to be included by the client and server applications" (Schofield, col. 6, lines 50-64). The concept of an object request broker is well known in the art and Schofield's code generator is not an object request broker as one of ordinary skill in the art would recognize.

Schofield also fails to anticipate <u>reading type information from a metadata repository</u>, wherein the type information is stored <u>in a database format</u> in the metadata repository. Instead, as described above, Schofield teaches the compiling of type specific client and server routines into implementation libraries. Nowhere does Schofield store type information in a database format in a metadata repository. There is no need in Schofield's system for reading type information from a metadata repository since the implementation libraries already contain type correct methods and routines.

Appellants further disagree with the Examiner's characterization of implementation library 81 from Fig. 4 as a metadata repository. Schofield teaches that the implementation library includes executable code such as the server applications and server stubs generated from the IDL specification of the object's interface (Schofield, column 5, lines 55-59). It is well understood that a library of executable code is not a

metadata repository. Schofield clearly describes how a compiler links the server application object code and the server stub object code to produce implementation library 81 (Schofield, col. 6, lines 47-49). Thus implementation library 81 contains server routines in executable format and cannot be a metadata repository from which type information is read and that stores the type information in a *database* format.

In the Advisory Action, the Examiner argues that Schofield uses client stub object code to produce a metadata repository. This is clearly the Examiner's own hindsight-based speculation since it is not mentioned at all by Schofield. Instead, Schofield teaches that client stub object code is linked to produce implementation library 71 and similarly server stub object code is linked to produce implementation library 81 (Schofield, column 6, lines 34-59). As argued above, implementation libraries 71 and 81 are libraries of executable routines and not metadata repositories.

Schofield also fails to teach translating the type information *from the database format to an interface definition language*. Firstly, as shown above, Schofield does not teach storing type information in a database format, but rather compiles header files for inclusion in client and server applications. Secondly, Schofield describes generating a CIN (ASCII based) descriptor ***from*** an IDL data type (Schofield, column 7, lines 20-22). In his Response to Arguments, the Examiner incorrectly states Appellants' argument. Appellants actually argue (as previously presented) that Schofield fails to teach translating the type information ***from*** *the database format* ***into*** *the interface definition language*. Instead, Schofield describes his method for generating a CIN (ASCII based) descriptor ***from*** an IDL data type (Schofield, column 7, lines 20-22). In fact, Schofield clearly illustrates this with an example where an ADD operation is translated from its IDL descriptor, "long Add (in long x, in long y);" to its equivalent CIN descriptor, "126861413+3+ADDA3+DFAFAF0+0+" (Schofield, column 10, lines 41-51). Hence, **Schofield teaches the opposite** of translating the type information ***into*** the interface definition language. In the Advisory Action, the Examiner again misrepresents Appellants' argument. The Examiner assumes that Appellants are arguing that Schofield

fails to teach "translating type information in IDL." However, as described above, Appellants are actually arguing that Schofield fails to teach translating type information from a database format into an interface definition language and that Schofield actually teaches the opposite. Thus, the Examiner's arguments regarding Schofield compiling IDL source files fails to rebut Appellants' argument.

Claim 26 is patentable over Schofield for at least the reasons given above regarding claim 22 from which it depends.

Claims 27 and 31:

Regarding claim 27, Schofield fails to anticipate a metadata gateway receiving the request for type information from the object request broker. Schofield makes no mention of any metadata gateway. Nor does Schofield describe a metadata gateway receiving a request for type information from an object request broker.

The Examiner argues that Schofield's server stub 89 is a metadata gateway. Appellants, however, disagree with such an interpretation of Schofield. Schofield teaches that "code generator 112 produces a client stub file 79 containing client stub functions and a server stub file 89 containing definitions for object implementations" (Schofield, col. 6, lines 30 – 32). Schofield does not teach that server stub 89 receives requests for type information from code generator 112, which the Examiner views as an object request broker. In fact, Schofield describes how code generator 112 creates server stub 89. Once Schofield's system is running (and requests are generated and sent) code generator 112 would not even be executing. Code generator 112 is only used to compile portions of client and server applications. Appellants fail to see how code generate 112 can be considered a metadata gateway that receives a request for type information from an object request broker.

The Examiner, in the Advisory Action of October 21, 2004, argues that server stub 89 receives a request for type information from code generator 112. However,

Schofield specifically describes how code generator 112 parses a PIF file and generates client and server stub files containing stub functions. Code generator 112 also creates a header that is included in the client and server stubs. Nowhere does Schofield mention code generator 112 sending a request for type information to a client stub. Furthermore, since code generator 112 is used only to create the sources files that will be compiled into server stub 89, and since code generator 112 and server stub 89 are never executed at the same time, there would be no way for code generator 112 to send a request to server stub 89.

Furthermore, the Examiner has incorrectly interpreted Schofield's code generator 112 as an object request broker. Schofield's code generator 112 is operable to generate files in the language of the client and server applications, and to produce a Compact IDL notation as illustrated in Figs. 5 – 7 and described starting at col. 7, line 20. Schofield does not teach that code generator 112 receives requests for type information as asserted by the Examiner. Schofield is very clear that code generator 112 "parses a source file line by line" and "produces an ASCII descriptor in the header file to be included by the client and server applications" (Schofield, col. 6, lines 50-64). The concept of an object request broker is well known in the art and Schofield's code generator is not an object request broker as one of ordinary skill in the art would recognize.

Schofield also fails to anticipate storing type information in a metadata repository, wherein the type information is stored in a database format in the metadata repository. Instead, as described above, Schofield teaches the compiling of type specific client and server routines into implementation libraries. Nothing in Schofield's system can be considered a metadata repository and nowhere does Schofield describe storing type information in a database format in a metadata repository.

Appellants further disagree with the Examiner's characterization of implementation library 81 from Fig. 4 as a metadata repository. Schofield teaches that the implementation library includes executable code such as the server applications and

server stubs generated from the IDL specification of the object's interface (Schofield, column 5, lines 55-59). It is well understood that a library of executable code is not a metadata repository. Schofield clearly describes how a compiler links the server application object code and the server stub object code to produce implementation library 81 (Schofield, col. 6, lines 47-49). Thus implementation library 81 contains server routines in executable format and cannot be a metadata repository from which type information is read and that stores the type information in a *database* format.

In the Advisory Action, the Examiner argues that Schofield uses client stub object code to produce a metadata repository. This is clearly the Examiner's own hindsight-based speculation since it is not mentioned at all by Schofield. Instead, Schofield teaches that client stub object code is linked to produce implementation library 71 and similarly server stub object code is linked to produce implementation library 81 (Schofield, column 6, lines 34-59). As argued above, implementation libraries 71 and 81 are libraries of executable routines and not metadata repositories.

Claim 31 is patentable over Schofield for at least the reasons presented above regarding claim 27 from which it depends.

Second Ground of Rejection:

Claims 2-4, 6, 11, 12, 17, 23-25 and 28-30 are finally rejected under 35 U.S.C. § 103(a) as being unpatentable over Schofield et al. in view of Kulkarni et al (U.S. Pat. No. 5,848,243 - hereinafter "Kulkarni"). Appellants traverse this rejection for at least the following reasons. Different groups of claims are addressed under their respective subheadings.

Claims 2 and 3:

Claims 2 and 3 are patentable over Schofield in view of Kulkarni for at least the reasons outlined above regarding claim 1, from which they depend.

In further regard to claim 2, Schofield in view of Kulkarni fails to teach or suggest translating the type information from the database format to an abstract syntax notation; and translating the type information from the abstract syntax notation to the interface definition language. As shown above regarding claim 1, Schofield fails to teach translating the type information from the database format to the interface definition language. Additionally, the Examiner admits that Schofield fails to teach wherein such translating includes translating the type information from the database format to an abstract syntax notation and translating the type information from the abstract syntax notation to the interface definition language. The Examiner relies upon Kulkarni and cites the Abstract as well as column 6, lines 20-43. The Examiner further argues that Kulkarni discloses using different data formats with the use of an abstract syntax notation. However, the portions of Kulkarni cited by the Examiner fail to teach or suggest the translation between these formats, contrary to the Examiner's assertion. Nowhere Kulkarni does not mention anything regarding translating anything between data formats. Instead, the Abstract describes how Kulkarni's system provides a common database for storing both a logical and physical layout of network resources (Kulkarni, column 2, lines 1-31).

Additionally, the Examiner's cited portion (column 6, lines 20-43) describes various views of a network using Kulkarni's system. Specifically, Kulkarni's system includes the ability to view network resources in various manners, both logical and physical and to setup and save new views of the network resources. Part of Kulkarni's system involves displaying the data formats and contents of various network nodes (Kulkarni, FIGs. 4A to 4C, and column 6, lines 20-26). Kulkarni does not teach translating type information between different formats. The only mention of an abstract syntax notation is at column 6, lines 41-44, where Kulkarni mentions that his internal network nodes (topoNodes) and user-defined views (topoViews) are preferably defined using GDMO format, which includes ASN.1. However, Kulkarni only refers to the software objects used in this display system as being defined using GDMO. Kulkarni

does not mention that any other network resources use GDMO format. Nor does Kulkarni disclose the translating of type information from a database format to an abstract syntax notation or translating of type information from the abstract syntax notation to the interface definition language. Furthermore, it would not make sense for Kulkarni to translate objects of classes internal to his network viewer program, to any other data format.

Thus, even if one were to combine Schofield with Kulkarni, as suggested by the Examiner, the resulting system would not result in a system that included translating the type information from the database format to an abstract syntax notation; and translating the type information from the abstract syntax notation to the interface definition language. Instead, modifying Schofield in view of Kulkarni would only result in a system that compiles client and server applications and implementation libraries from IDL source files, as taught by Schofield, but that also includes a viewer application capable of displaying both logical and physical relationships between network resources, as taught by Kulkarni.

Claim 4:

Claim 4 is patentable over Schofield in view of Kulkarni for at least the reasons set forth above regarding claim 1 from which claim 4 depends. Furthermore, Schofield in view of Kulkarni fails to teach or suggest translating the type information from the abstract syntax notation to an object specification language; and translating the type information from the object specification language to the interface definition language.

As shown above regarding claim 1, Schofield fails to teach translating the type information from the database format to the interface definition language. The Examiner relies upon Kulkarni and cites the Abstract as well as column 6, lines 20-43. The Examiner further argues that Kulkarni discloses using different data formats with the use of an abstract syntax notation. However, the portions of Kulkarni cited by the Examiner fail to teach or suggest the translation between these formats, contrary to the Examiner's

assertion. Nowhere Kulkarni does not mention anything regarding translating anything between data formats. Instead, the Abstract describes how Kulkarni's system provides a common database for storing both a logical and physical layout of network resources (Kulkarni, column 2, lines 1-31).

Additionally, the Examiner's cited portion (column 6, lines 20-43) describes various views of a network using Kulkarni's system. Specifically, Kulkarni's system includes the ability to view network resources in various manners, both logical and physical and to setup and save new views of the network resources. Part of Kulkarni's system involves displaying the data formats and contents of various network nodes (Kulkarni, FIGs. 4A to 4C, and column 6, lines 20-26). Kulkarni does not teach translating type information between different formats. The only mention of an abstract syntax notation is at column 6, lines 41-44, where Kulkarni mentions that his internal network nodes (topoNodes) and user-defined views (topoViews) are preferably defined using GDMO format, which includes ASN.1. However, Kulkarni only refers to the software objects used in this display system as being defined using GDMO. Kulkarni does not mention that any other network resources use GDMO format. Nor does Kulkarni disclose the translating of type information from a database format to an abstract syntax notation or translating of type information from the abstract syntax notation to the interface definition language. Furthermore, it would not make sense for Kulkarni to translate objects of classes internal to his network viewer program, to any other data format.

Thus, even if one were to combine Schofield with Kulkarni, as suggested by the Examiner, the resulting system would not result in a system that included translating the type information from the abstract syntax notation to an object specification language; and translating the type information from the object specification language to the interface definition language. Instead, modifying Schofield in view of Kulkarni would result in a system that compiles client and server applications and implementation libraries from IDL source files, as taught by Schofield, but that also includes a viewer

application capable of displaying both logical and physical relationships between network resources, as taught by Kulkarni.

Claim 6:

Claim 6 is patentable over Schofield in view of Kulkarni for at least the reasons given above regarding claims 1 and 5 from which it depends.

Claim 11:

Claim 11 is patentable over Schofield in view of Kulkarni for at least the reasons above regarding claim 10 from which it depends. Additionally, Schofield in view of Kulkarni fails to teach or suggest translating the type information from the interface definition language to an abstract syntax notation; and translating the type information from the abstract syntax notation to the database format. As shown above regarding claim 10, Schofield fails to teach translating the type information from the interface definition language to a database format. The Examiner further argues that Kulkarni discloses using different data formats with the use of an abstract syntax notation and cites the Abstract as well as column 6, lines 20-43 of Kulkarni. However, the portions of Kulkarni cited by the Examiner fail to teach or suggest the translation between these formats, contrary to the Examiner's assertion. Nowhere does Kulkarni mention translating anything between data formats. Instead, Kulkarni's system provides a common database for storing both a logical and physical layout of network resources (Kulkarni, Abstract and column 2, lines 1-31).

Additionally, the Examiner's cited portion (column 6, lines 20-43) describes various views of a network using Kulkarni's system. Specifically, Kulkarni's system includes the ability to view network resources in various manners, both logical and physical and to setup and save new views of the network resources. Part of Kulkarni's system involves displaying the data formats and contents of various network nodes (Kulkarni, FIGs. 4A to 4C, and column 6, lines 20-26). Kulkarni does not teach

translating type information between different formats. The only mention of an abstract syntax notation is at column 6, lines 41-44, where Kulkarni mentions that his internal network nodes (topoNodes) and user-defined views (topoViews) are preferably defined using GDMO format, which includes ASN.1. However, Kulkarni only refers to the software objects used in this display system as being defined using GDMO. Kulkarni does not mention that any other network resources use GDMO format. Nor does Kulkarni disclose the translating of type information from a database format to an abstract syntax notation or translating of type information from the abstract syntax notation to the interface definition language. Furthermore, it would not make sense for Kulkarni to translate objects of classes internal to his network viewer program, to any other data format.

Thus, even if one were to combine Schofield with Kulkarni, as suggested by the Examiner, the resulting system would not result in a system that included translating the type information from the interface definition language to an abstract syntax notation; and translating the type information from the abstract syntax notation to the database format. Instead, modifying Schofield in view of Kulkarni would only result in a system that compiles client and server applications and implementation libraries from IDL source files, as taught by Schofield, but that also includes a viewer application capable of displaying both logical and physical relationships between network resources, as taught by Kulkarni.

Claim 12:

Claim 12 is patentable over Schofield in view of Kulkarni for at least the reasons given above regarding claims 10 and 11 from which it depends. Additionally, Schofield in view of Kulkarni fails to teach or suggest translating the type information from the interface definition language to an object specification language; and translating the type information from the object specification language to the abstract syntax notation. As shown above regarding claim 10, Schofield fails to teach translating the type information from the interface definition language to a database format. The Examiner argues that

Kulkarni discloses using different data formats with the use of an abstract syntax notation, citing the Abstract as well as column 6, lines 20-43 of Kulkarni. However, the cited portions of Kulkarni fail to teach or suggest the translation between these formats, contrary to the Examiner's assertion. Nowhere does Kulkarni mention translating anything between data formats. Instead, the Abstract describes how Kulkarni's system provides a common database for storing both a logical and physical layout of network resources (Kulkarni, column 2, lines 1-31).

Additionally, the Examiner's cited portion (column 6, lines 20-43) describes various views of a network using Kulkarni's system. Specifically, Kulkarni's system includes the ability to view network resources in various manners, both logical and physical and to setup and save new views of the network resources. Part of Kulkarni's system involves displaying the data formats and contents of various network nodes (Kulkarni, FIGs. 4A to 4C, and column 6, lines 20-26). Kulkarni does not teach translating type information between different formats. The only mention of an abstract syntax notation is at column 6, lines 41-44, where Kulkarni mentions that his internal network nodes (topoNodes) and user-defined views (topoViews) are preferably defined using GDMO format, which includes ASN.1. However, Kulkarni only refers to the software objects used in this display system as being defined using GDMO. Kulkarni does not mention that any other network resources use GDMO format. Nor does Kulkarni disclose the translating of type information from a database format to an abstract syntax notation or translating of type information from the abstract syntax notation to the interface definition language. Furthermore, it would not make sense for Kulkarni to translate objects of classes internal to his network viewer program, to any other data format.

Thus, even if one were to combine Schofield with Kulkarni, as suggested by the Examiner, the resulting system would not result in a system that included translating the type information from the interface definition language to an object specification language; and translating the type information from the object specification language to

the abstract syntax notation. Instead, modifying Schofield in view of Kulkarni would only result in a system that compiles client and server applications and implementation libraries from IDL source files, as taught by Schofield, but that also includes a viewer application capable of displaying both logical and physical relationships between network resources, as taught by Kulkarni.

<u>Claim 17</u>:

Claim 17 is patentable over Schofield in view of Kulkarni for at least the reasons given above regarding claim 14 from which it depends. Furthermore, Schofield in view of Kulkarni fails to teach or suggest a metadata gateway that comprises: a <u>library of data types expressed in an abstract syntax notation</u>, wherein the abstract syntax notation comprises a metadata notation language; <u>a plurality of object types</u>, wherein each object type comprises one or more of the data types from the library of data types; and an interface to the plurality of object types, wherein the <u>interface is operable to provide one or more clients with access to the metadata</u> as expressed in the interface definition language.

The Examiner has rejected claim 17, but has failed to provide any reason for his rejection and has failed to cite any portion of Schofield or Kulkarni, either singly or in combination, that teaches or suggests the limitations recited in claim 17. Specifically, in the rejection of claim 17, the Examiner only argues that Schofield in view Kulkarni teach translating type information between various formats. (Please refer to Appellants' arguments above regarding claims 2, 4, 11, and 12, for a refutation of the Examiner's assertion.)

Furthermore, the Examiner has repeated argued that Schofield's server stub 89 is a metadata gateway, but Schofield fails to describe server stub 89 as including a library of data types expressed in an abstract syntax notation including a metadata notation language. Instead, Schofield described how both client stub 79 and server stub 89 contain stub functions and definitions for object implementation. Schofield does not mention the

user of abstract syntax notation. Schofield further fails to describe server stub 89 as including a plurality of object types or as including an interface to the plurality of object types, wherein the interface is operable to provide clients with access to the metadata as expressed in the interface definition language.

Furthermore, Kulkarni also does not mention anything about a metadata gateway comprising a library of data types, a plurality of data objects, and an interface to the plurality of object types. Thus, Schofield in view of Kulkarni fails to teach the limitations recited in claim 17.

Claims 23 and 24:

Claims 23 and 24 are patentable over Schofield in view of Kulkarni for at least the reasons outlined above regarding claim 22, from which they depend.

In further regard to claim 23, Schofield in view of Kulkarni fails to teach or suggest translating the type information from the database format to an abstract syntax notation; and translating the type information from the abstract syntax notation to the interface definition language. As shown above regarding claim 22, Schofield fails to teach translating the type information from the database format to the interface definition language. Additionally, the Examiner admits that Schofield fails to teach wherein such translating includes translating the type information from the database format to an abstract syntax notation and translating the type information from the abstract syntax notation to the interface definition language. The Examiner relies upon Kulkarni and cites the Abstract as well as column 6, lines 20-43. The Examiner further argues that Kulkarni discloses using different data formats with the use of an abstract syntax notation. However, the portions of Kulkarni cited by the Examiner fail to teach or suggest the translation between these formats, contrary to the Examiner's assertion. Nowhere Kulkarni does not mention anything regarding translating anything between data formats. Instead, the Abstract describes how Kulkarni's system provides a common

database for storing both a logical and physical layout of network resources (Kulkarni, column 2, lines 1-31).

Additionally, the Examiner's cited portion (column 6, lines 20-43) describes various views of a network using Kulkarni's system. Specifically, Kulkarni's system includes the ability to view network resources in various manners, both logical and physical and to setup and save new views of the network resources. Part of Kulkarni's system involves displaying the data formats and contents of various network nodes (Kulkarni, FIGs. 4A to 4C, and column 6, lines 20-26). Kulkarni does not teach translating type information between different formats. The only mention of an abstract syntax notation is at column 6, lines 41-44, where Kulkarni mentions that his internal network nodes (topoNodes) and user-defined views (topoViews) are preferably defined using GDMO format, which includes ASN.1. However, Kulkarni only refers to the software objects used in this display system as being defined using GDMO. Kulkarni does not mention that any other network resources use GDMO format. Nor does Kulkarni disclose the translating of type information from a database format to an abstract syntax notation or translating of type information from the abstract syntax notation to the interface definition language. Furthermore, it would not make sense for Kulkarni to translate objects of classes internal to his network viewer program, to any other data format.

Thus, even if one were to combine Schofield with Kulkarni, as suggested by the Examiner, the resulting system would not result in a system that included translating the type information from the database format to an abstract syntax notation; and translating the type information from the abstract syntax notation to the interface definition language. Instead, modifying Schofield in view of Kulkarni would only result in a system that compiles client and server applications and implementation libraries from IDL source files, as taught by Schofield, but that also includes a viewer application capable of displaying both logical and physical relationships between network resources, as taught by Kulkarni.

Claim 25:

Claim 25 is patentable over Schofield in view of Kulkarni for at least the reasons set forth above regarding claim 22 from which it depends. Furthermore, Schofield in view of Kulkarni fails to teach or suggest translating the type information from the abstract syntax notation to an object specification language; and translating the type information from the object specification language to the interface definition language.

As shown above regarding claim 22, Schofield fails to teach translating the type information from the database format to the interface definition language. The Examiner relies upon Kulkarni and cites the Abstract as well as column 6, lines 20-43. The Examiner further argues that Kulkarni discloses using different data formats with the use of an abstract syntax notation. However, the portions of Kulkarni cited by the Examiner fail to teach or suggest the translation between these formats, contrary to the Examiner's assertion. Nowhere Kulkarni does not mention anything regarding translating anything between data formats. Instead, the Abstract describes how Kulkarni's system provides a common database for storing both a logical and physical layout of network resources (Kulkarni, column 2, lines 1-31).

Additionally, the Examiner's cited portion (column 6, lines 20-43) describes various views of a network using Kulkarni's system. Specifically, Kulkarni's system includes the ability to view network resources in various manners, both logical and physical and to setup and save new views of the network resources. Part of Kulkarni's system involves displaying the data formats and contents of various network nodes (Kulkarni, FIGs. 4A to 4C, and column 6, lines 20-26). Kulkarni does not teach translating type information between different formats. The only mention of an abstract syntax notation is at column 6, lines 41-44, where Kulkarni mentions that his internal network nodes (topoNodes) and user-defined views (topoViews) are preferably defined using GDMO format, which includes ASN.1. However, Kulkarni only refers to the

software objects used in this display system as being defined using GDMO. Kulkarni does not mention that any other network resources use GDMO format. Nor does Kulkarni disclose the translating of type information from a database format to an abstract syntax notation or translating of type information from the abstract syntax notation to the interface definition language. Furthermore, it would not make sense for Kulkarni to translate objects of classes internal to his network viewer program, to any other data format.

Thus, even if one were to combine Schofield with Kulkarni, as suggested by the Examiner, the resulting system would not result in a system that included translating the type information from the abstract syntax notation to an object specification language; and translating the type information from the object specification language to the interface definition language. Instead, modifying Schofield in view of Kulkarni would result in a system that compiles client and server applications and implementation libraries from IDL source files, as taught by Schofield, but that also includes a viewer application capable of displaying both logical and physical relationships between network resources, as taught by Kulkarni.

<u>Claims 28 and 29</u>:

Claims 28 and 29 are patentable over Schofield in view of Kulkarni for at least the reasons above regarding claim 27 from which they depend.

Additionally, regarding claim 28, Schofield in view of Kulkarni fails to teach or suggest translating the type information <u>from the interface definition language to an abstract syntax notation</u>; and translating the type information <u>from the abstract syntax notation to the database format</u>. The Examiner argues that Kulkarni discloses using different data formats with the use of an abstract syntax notation and cites the Abstract as well as column 6, lines 20-43 of Kulkarni. However, the portions of Kulkarni cited by the Examiner fail to teach or suggest the translation between these formats, contrary to

the Examiner's assertion. Nowhere does Kulkarni mention translating anything between data formats. Instead, Kulkarni's system provides a common database for storing both a logical and physical layout of network resources (Kulkarni, Abstract and column 2, lines 1-31).

Additionally, the Examiner's cited portion (column 6, lines 20-43) describes various views of a network using Kulkarni's system. Specifically, Kulkarni's system includes the ability to view network resources in various manners, both logical and physical and to setup and save new views of the network resources. Part of Kulkarni's system involves displaying the data formats and contents of various network nodes (Kulkarni, FIGs. 4A to 4C, and column 6, lines 20-26). Kulkarni does not teach translating type information between different formats. The only mention of an abstract syntax notation is at column 6, lines 41-44, where Kulkarni mentions that his internal network nodes (topoNodes) and user-defined views (topoViews) are preferably defined using GDMO format, which includes ASN.1. However, Kulkarni only refers to the software objects used in this display system as being defined using GDMO. Kulkarni does not mention that any other network resources use GDMO format. Nor does Kulkarni disclose the translating of type information from a database format to an abstract syntax notation or translating of type information from the abstract syntax notation to the interface definition language. Furthermore, it would not make sense for Kulkarni to translate objects of classes internal to his network viewer program, to any other data format.

Thus, even if one were to combine Schofield with Kulkarni, as suggested by the Examiner, the resulting system would not result in a system that included translating the type information from the interface definition language to an abstract syntax notation; and translating the type information from the abstract syntax notation to the database format. Instead, modifying Schofield in view of Kulkarni would only result in a system that compiles client and server applications and implementation libraries from IDL source files, as taught by Schofield, but that also includes a viewer application capable of

displaying both logical and physical relationships between network resources, as taught by Kulkarni.

<u>Claim 30</u>:

Claim 30 is patentable over Schofield in view of Kulkarni for at least the reasons presented above regarding claim 27 from which it depends. Additionally, Schofield in view of Kulkarni fails to teach or suggest translating the type information <u>from the interface definition language to an object specification language</u>; and translating the type information <u>from the object specification language to the abstract syntax notation</u>. The Examiner argues that Kulkarni discloses using different data formats with the use of an abstract syntax notation, citing the Abstract as well as column 6, lines 20-43 of Kulkarni. However, the cited portions of Kulkarni fail to teach or suggest the translation between these formats, contrary to the Examiner's assertion. Nowhere does Kulkarni mention translating anything between data formats. Instead, the Abstract describes how Kulkarni's system provides a common database for storing both a logical and physical layout of network resources (Kulkarni, column 2, lines 1-31).

Additionally, the Examiner's cited portion (column 6, lines 20-43) describes various views of a network using Kulkarni's system. Specifically, Kulkarni's system includes the ability to view network resources in various manners, both logical and physical and to setup and save new views of the network resources. Part of Kulkarni's system involves displaying the data formats and contents of various network nodes (Kulkarni, FIGs. 4A to 4C, and column 6, lines 20-26). Kulkarni does not teach translating type information between different formats. The only mention of an abstract syntax notation is at column 6, lines 41-44, where Kulkarni mentions that his internal network nodes (topoNodes) and user-defined views (topoViews) are preferably defined using GDMO format, which includes ASN.1. However, Kulkarni only refers to the software objects used in this display system as being defined using GDMO. Kulkarni does not mention that any other network resources use GDMO format. Nor does Kulkarni disclose the translating of type information from a database format to an abstract

syntax notation or translating of type information from the abstract syntax notation to the interface definition language. Furthermore, it would not make sense for Kulkarni to translate objects of classes internal to his network viewer program, to any other data format.

Thus, even if one were to combine Schofield with Kulkarni, as suggested by the Examiner, the resulting system would not result in a system that included translating the type information from the interface definition language to an object specification language; and translating the type information from the object specification language to the abstract syntax notation. Instead, modifying Schofield in view of Kulkarni would only result in a system that compiles client and server applications and implementation libraries from IDL source files, as taught by Schofield, but that also includes a viewer application capable of displaying both logical and physical relationships between network resources, as taught by Kulkarni.

VIII. CONCLUSION

For the foregoing reasons, it is submitted that the Examiner's rejection of claims 1-31 was erroneous, and reversal of his decision is respectfully requested.

The Commissioner is authorized to charge the appeal brief fee of $500.00 and any other fees that may be due to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5181-46200/RCK. This Appeal Brief is submitted with a return receipt postcard.

Respectfully submitted,

Robert C. Kowert
Reg. No. 39,255
Attorney for Appellants

Meyertons, Hood, Kivlin, Kowert & Goetzel, P.C.

P.O. Box 398
Austin, TX 78767-0398
(512) 853-8850

Date: December 21, 2004

IX. CLAIMS APPENDIX

The claims on appeal are as follows.

1. A method for managing a network, the method comprising:

a client generating a request for type information for an attribute or event, wherein the request is expressed in an interface definition language, wherein the interface definition language is operable to define object interfaces across a plurality of platforms and across a plurality of programming languages;

sending the request for type information to an object request broker;

a metadata gateway receiving the request for type information from the object request broker;

reading the type information from a metadata repository, wherein the type information is stored in a database format in the metadata repository;

translating the type information from the database format to the interface definition language;

the metadata gateway sending the translated type information to the object request broker; and

the client receiving the translated type information for the attribute or event through the object request broker, wherein the translated type information is expressed in the interface definition language.

2. The method of claim 1, wherein the translating the type information from the database format to the interface definition language comprises:

translating the type information from the database format to an abstract syntax notation; and

translating the type information from the abstract syntax notation to the interface definition language.

3. The method of claim 2, wherein the abstract syntax notation is Abstract Syntax Notation One (ASN1).

4. The method of claim 2, wherein the translating the type information from the abstract syntax notation to the interface definition language comprises:

translating the type information from the abstract syntax notation to an object specification language; and

translating the type information from the object specification language to the interface definition language.

5. The method of claim 1, wherein the sending the request for type information to an object request broker, the metadata gateway receiving the request for type information from the object request broker, the metadata gateway sending the translated type information to the object request broker, and the client receiving the translated type information for the attribute or event through the object request broker are effected via an internet inter-object communication protocol.

6. The method of claim 5, wherein the internet inter-object communication protocol comprises Internet Inter-Object Protocol (IIOP).

7. The method of claim 1, wherein the metadata gateway is implemented on a single server computer system.

8. The method of claim 1, wherein the metadata gateway is distributed over a plurality of servers, wherein each of the plurality of servers presents a substantially identical view of the metadata gateway.

9. The method of claim 1, wherein the interface definition language is class-independent.

10. A method for managing a network, the method comprising:

a client generating a request to encode type information for an object, attribute, or event, wherein the request is expressed in an interface definition language, wherein the interface definition language is operable to define object interfaces across a plurality of platforms and across a plurality of programming languages;

sending the request to an object request broker;

a metadata gateway receiving the request to encode the type information from the object request broker;

translating the type information from the interface definition language to a database format; and

storing the type information in a metadata repository, wherein the type information is stored in a database format in the metadata repository.

11. The method of claim 10, wherein the translating the type information from the interface definition language to the database format comprises:

translating the type information from the interface definition language to an abstract syntax notation; and

translating the type information from the abstract syntax notation to the database format.

12. The method of claim 11, wherein the translating the type information from the interface definition language to the abstract syntax notation comprises:

translating the type information from the interface definition language to an object specification language; and

translating the type information from the object specification language to the abstract syntax notation.

13. The method of claim 10, wherein the sending the request to an object request broker and the metadata gateway receiving the request to encode the type information from the object request broker are effected via an internet inter-object communication protocol.

14. A network management system comprising:

a metadata repository, wherein the metadata repository comprises metadata concerning object classes for a plurality of managed objects, wherein the metadata comprises information expressed in a database format, and wherein the managed objects correspond to managed devices on a network; and

a metadata gateway which is communicatively coupled to the metadata repository and to an object request broker, wherein the metadata gateway is operable to send and receive the metadata from the database, wherein the metadata gateway provides translation of the metadata to and from the database format and an interface definition language, wherein the interface definition language is operable to define object interfaces across a plurality of platforms and across a plurality of programming languages.

15. The network management system of claim 14, wherein the managed devices comprise a telephone system.

16. The network management system of claim 14, wherein the managed devices comprise a network switch.

17. The network management system of claim 14, wherein the metadata gateway further comprises:

a library of data types expressed in an abstract syntax notation, wherein the abstract syntax notation comprises a metadata notation language;

a plurality of object types, wherein each object type comprises one or more of the data types from the library of data types; and

an interface to the plurality of object types, wherein the interface is operable to provide one or more clients with access to the metadata as expressed in the interface definition language.

18. The network management system of claim 17, wherein the interface to the plurality of object types is a programming-language-independent and platform-independent interface.

19. The network management system of claim 17, wherein the plurality of object types comprise CORBA objects.

20. The network management system of claim 14, wherein the object request broker is configurable to be accessed by a plurality of network management clients to obtain the metadata as expressed in the generic interface.

21. The network management system of claim 14, wherein the object request broker comprises a CORBA ORB.

22. A carrier medium comprising program instructions, wherein the program instructions are computer-executable to implement:

a metadata gateway receiving a request for type information from an object request broker;

reading the type information from a metadata repository, wherein the type information is stored in a database format in the metadata repository;

translating the type information from the database format to an interface definition language; and

the metadata gateway sending the translated type information to the object request broker.

23. The carrier medium of claim 22, wherein in translating the type information from the database format to the interface definition language, the program instructions are further computer-executable to implement:

translating the type information from the database format to an abstract syntax notation; and

translating the type information from the abstract syntax notation to the interface definition language.

24. The carrier medium of claim 23, wherein the abstract syntax notation is Abstract Syntax Notation One (ASN1).

25. The carrier medium of claim 22, wherein in translating the type information from the abstract syntax notation to the interface definition language, the program instructions are further computer-executable to implement:

translating the type information from the abstract syntax notation to an object specification language; and

translating the type information from the object specification language to the interface definition language.

26. The carrier medium of claim 22, wherein the interface definition language is class-independent.

27. A carrier medium comprising program instructions which are computer-executable to implement:

a metadata gateway receiving a request to encode type information from an object request broker;

translating the type information from an interface definition language to a database format; and

storing the type information in a metadata repository, wherein the type information is stored in a database format in the metadata repository.

28. The carrier medium of claim 27, wherein in translating the type information from the interface definition language to the database format, the program instructions are further computer-executable to implement:

translating the type information from the interface definition language to an abstract syntax notation; and

translating the type information from the abstract syntax notation to the database format.

29. The carrier medium of claim 28, wherein the abstract syntax notation is Abstract Syntax Notation One (ASN1).

30. The carrier medium of claim 27, wherein in translating the type information from the interface definition language to the abstract syntax notation, the program instructions are further computer-executable to implement:

translating the type information from the interface definition language to an object specification language; and

translating the type information from the object specification language to the abstract syntax notation.

31. The carrier medium of claim 27, wherein the interface definition language is class-independent.

X. EVIDENCE APPENDIX

No evidence submitted under 37 CFR §§ 1.130, 1.131 or 1.132 or otherwise entered by the Examiner is relied upon in this appeal.

XI. RELATED PROCEEDINGS APPENDIX

There are no related proceedings.